Compressed Tree Canonization

Compressed Tree Canonization

Markus Lohrey[1], Sebastian Maneth[2], and Fabian Peternek[2]

[1] Universität Siegen, Germany
[2] University of Edinburgh, UK

Abstract. Straight-line (linear) context-free tree (SLT) grammars have been used to compactly represent ordered trees. Equivalence of SLT grammars is decidable in polynomial time. Here we extend this result and show that isomorphism of unordered trees given as SLT grammars is decidable in polynomial time. The result generalizes to isomorphism of unrooted trees and bisimulation equivalence. For non-linear SLT grammars which can have double-exponential compression ratios, we prove that unordered isomorphism and bisimulation equivalence are PSPACE-hard and in EXPTIME.

1 Introduction

Deciding isomorphism between various mathematical objects is an important topic in theoretical computer science that has led to intriguing open problems like the precise complexity of the graph isomorphism problem. An example of an isomorphism problem, where the knowledge seems to be rather complete, is tree isomorphism. Aho, Hopcroft and Ullman [1, page 84] proved that isomorphism of unordered trees (rooted or unrooted) can be decided in linear time. An unordered tree is a tree, where the children of a node are not ordered. The precise complexity of tree isomorphism was finally settled by Lindell [11], Buss [5], and Jenner et al. [9]: tree isomorphism is LOGSPACE-complete if the trees are represented by pointer structures [11,9] and ALOGTIME-complete if the trees are represented by expressions [5,9]. All these results deal with trees that are given explicitly (either by an expression or a pointer structure). In this paper, we deal with the isomorphism problem for trees that are given in a succinct way. Several succinct encoding schemes for graphs exist in the literature. Galperin and Wigderson [8] considered graphs that are given by a Boolean circuit for the adjacency matrix. Subsequent work showed that the complexity of a problem undergoes an exponential jump when going from the standard input representation to the circuit representation; this phenomenon is known as upgrading, see [7] for more details and references. Concerning graph isomorphism, it was shown in [7] that its succinct version is PSPACE-hard, even for very restricted classes of Boolean circuits (DNFs and CNFs).

In this paper, we consider another succinct input representation that has turned out to be more amenable to efficient algorithms, and, in particular, does not show the upgrading phenomenon known for Boolean circuits: straight-line context-free grammars, i.e., context-free grammars that produce a single object. Such grammars have been intensively studied for strings and recently also for trees. Using a straight-line grammar, repeated patterns in an string or tree can be abbreviated by a nonterminal which can

be used in different contexts. For strings, this idea is known as grammar-based compression [6,12], and it was extended to trees in [4]. In fact this approach can be also extended to general graphs by using hyperedge replacement graph grammars; the resulting formalism is known as hierarchical graph representation [10].

The main topic of this paper is the isomorphism problem for trees that are succinctly represented by straight-line context-free tree grammars (ST *grammars*). An example of such a grammar contains the productions $S \to A_0(a)$, $A_i(y) \to A_{i+1}(A_{i+1}(y))$ for $0 \leq i \leq n-1$, and $A_n(y) \to f(y,y)$ (here y is called a parameter and in general several parameters may occur in a rule). This grammar produces a full binary tree of height 2^n and hence has $2^{2^n+1}-1$ many nodes. Thus, an ST grammar may produce a tree, whose size is doubly exponential in the size of the grammar. The reason for this double exponential blow-up is copying: the parameter y occurs twice in the right-hand side of the production $A_n(y) \to f(y,y)$. If this is not allowed, i.e., if every parameter occurs at most once in every right-hand side, then the grammar is a *straight-line linear context-free tree grammar* (SLT *grammar*). The latter generalize dags (directed acyclic graphs) that allow to share repeated subtrees of a tree, whereas SLT grammars can also share repeated patterns that are not complete subtrees.

Several algorithmic problems are harder for trees represented by ST grammars than trees represented by SLT grammars. A good example is the membership problem for tree automata (PTIME-complete for SLT grammars and PSPACE-complete for ST grammars, see [12, Theorem 39]). A similar situation arises for the isomorphism problem: we prove that the isomorphism problem for (rooted or unrooted) unordered trees that are given by SLT grammars (resp., ST grammars) is PTIME-complete (resp., PSPACE-hard and in EXPTIME). Our polynomial time algorithm for SLT grammars constructs from a given SLT grammar G a new SLT grammar G' that produces a canonical representation (based on lexicographic ordering of depth-first left-to-right traversals) of the tree produced by G. For unrooted SLT-compressed trees, we first compute a compressed representation of the center node of a given SLT-compressed unrooted tree t. Then we compute an SLT grammar that produces the rooted version of t that is rooted in the center node. This is also the standard reduction of the unrooted isomorphism problem to the rooted isomorphism problem in the uncompressed setting, but it requires some work to carry out this reduction in polynomial time in the SLT-compressed setting.

Our techniques can be also used to show that checking bisimulation equivalence of trees that are represented by SLT grammars is PTIME-complete. This generalizes the well-known PTIME-completeness of bisimulation for dags [2]. In this context, it is interesting to note that bisimulation equivalence for graphs that are given by hierarchical graph representations is PSPACE-hard and in EXPTIME [3].

Full proofs can be found in the long version [13].

2 Preliminaries

For $k \geq 0$ let $[k] = \{1, \ldots, k\}$. Let Σ be an alphabet. By T_Σ we denote the set of all (ordered, rooted) trees over the alphabet Σ. It is defined recursively as the smallest set of strings such that if $t_1, \ldots, t_k \in T_\Sigma$ and $k \geq 0$ then also $\sigma(t_1, \ldots, t_k) \in T_\Sigma$. For the tree $a()$ we simply write a. The set $D(t)$ of *Dewey addresses* of a tree $t = \sigma(t_1, \ldots, t_k)$

is the subset of $\mathbb{N}^*$ defined recursively as $\{\varepsilon\} \cup \bigcup_{i\in[k]} i \cdot D(t_i)$. Thus ε is the root node of t and $u \cdot i$ is the i-th child of u. For $u \in D(t)$, we denote by $t[u] \in \Sigma$ the symbol at u, i.e., if $t = \sigma(t_1, \ldots, t_k)$, then $t[\varepsilon] = \sigma$ and $t[i \cdot u] = t_i[u]$. The *size* of t is $|t| = |D(t)|$.

A *ranked alphabet* N is a finite set of symbols each of which equipped with a non-negative integer, called its "rank". We write $N^{(k)}$ for the set of symbols in N that have rank k. For an alphabet Σ and a ranked alphabet N, we denote by $T_{N\cup\Sigma}$ the set of trees t over $N \cup \Sigma$ with the property that if $t[u] = A \in N^{(k)}$, then $u \cdot i \in D(t)$ if and only if $i \in [k]$. Thus, if a node is labeled by a ranked symbol, then the rank determines the number of children of the node. We fix a set $Y = \{y_1, y_2, \ldots\}$ of *parameters*, which are symbols of rank 0. For y_1 we also write y. We write $T_{\Sigma\cup N}(Y)$ for $T_{\Sigma\cup N\cup Y}$. For trees $t, t_1, \ldots, t_k \in T_{\Sigma\cup N}(Y)$ we denote by $t[y_j \leftarrow t_j \mid j \in [k]]$ the tree obtained from t by replacing in parallel every occurrence of y_j ($j \in [k]$) by t_j. A *context* is a tree in $T_{\Sigma\cup N}(\{y\})$ with exactly one occurrence of y. Let $\mathcal{C}_{\Sigma\cup N}$ be the set of all contexts and let $\mathcal{C}_\Sigma = \mathcal{C}_{\Sigma\cup N} \cap T_\Sigma(\{y\})$. For a context $t(y)$ and a tree t' we write $t[t']$ for $t[y \leftarrow t']$.

A *context-free tree grammar* is a tuple $G = (N, \Sigma, S, P)$ where N is a ranked alphabet of nonterminal symbols, Σ is an alphabet of terminal symbols with $\Sigma \cap N = \emptyset$, $S \in N^{(0)}$ is the start nonterminal, and P is a finite set of productions of the form $A(y_1, \ldots, y_k) \to t$ where $A \in N^{(k)}$, $k \geq 0$, and $t \in T_{N\cup\Sigma}(\{y_1, \ldots, y_k\})$. Occasionally, we consider context-free tree grammars without a start nonterminal. Two trees $\xi, \xi' \in T_{N\cup\Sigma}(Y)$ are in the one-step derivation relation $\Rightarrow_G$ induced by G, if ξ has a subtree $A(t_1, \ldots, t_k)$ with $A \in N^{(k)}$, $k \geq 0$ such that ξ' is obtained from ξ by replacing this subtree by $t[y_j \leftarrow t_j \mid j \in [k]]$, where $A(y_1, \ldots, y_k) \to t$ is a production in P. The tree language $L(G)$ produced by G is $\{t \in T_\Sigma \mid S \Rightarrow_G^* t\}$. The *size* of the grammar G is $|G| = \sum_{(A(y_1,\ldots,y_k)\to t)\in P} |t|$. The grammar $G = (N, \Sigma, S, P)$ is *deterministic* if for every $A \in N$ there is exactly one production of the form $A \to t$. The grammar G is *acyclic*, if there is a linear order $<$ on N such that $A < B$ whenever B occurs in a tree t with $(A \to t) \in P$. A deterministic and acyclic grammar is called *straight-line*. Note that $|L(G)| = 1$ for a straight-line grammar. We denote the unique tree t produced by the straight-line tree grammar G by $\mathsf{val}(G)$. Moreover, for a tree $t \in T_{\Sigma\cup N}(Y)$ we denote with $\mathsf{val}_G(t)$ the unique tree from $T_\Sigma(Y)$ such that $t \Rightarrow_G^* \mathsf{val}_G(t)$. If G is clear from the context, we simply write $\mathsf{val}(t)$ for $\mathsf{val}_G(t)$. The grammar G is *linear* if for every production $(A \to t) \in P$ and every $y \in Y$, y occurs at most once in t.

For a straight-line context-free tree grammar (resp., straight-line linear context-free tree grammar) we say ST *grammar* (resp.. SLT *grammar*.) Occasionally, we also consider SLT grammars, where the start nonterminal belongs to $N^{(1)}$, i.e., has rank 1. For such a 1-SLT *grammar* G it holds that $\mathsf{val}(G) \in \mathcal{C}_\Sigma$. Most of this paper is about SLT grammars, only at the very end of the paper we consider general ST grammars. SLT grammars generalize rooted node-labelled dags (directed acyclic graph), where the tree defined by such a dag is obtained by unfolding the dag starting from the root (formally, the nodes of the tree are the directed paths in the dag that start in the root). A dag can be viewed as an SLT grammar, where all nonterminals have rank 0 (the nodes of the dag correspond to the nonterminal of the SLT grammar). Dags are less succinct than SLT grammars (take the tree $f^N(a)$ for $N = 2^n$), which in turn are less succinct than general ST grammars (take a full binary tree of height 2^n).

In the literature, SLT grammars are usually defined over ranked terminal alphabets. The proof of the following result from [14] also works for an unranked alphabet Σ.

Lemma 1. *One can transform in polynomial time an* SLT *grammar into an equivalent* SLT *grammar, where every nonterminal has rank at most one and each production has one of the following four types (where $\sigma \in \Sigma$ and $A, B, C, A_1, \ldots, A_k \in N$):*

(1) $A \to \sigma(A_1, \ldots, A_k)$,
(2) $A \to B(C)$,
(3) $A(y) \to \sigma(A_1, \ldots, A_i, y, A_{i+1}, \ldots, A_k)$, *or*
(4) $A(y) \to B(C(y))$.

In the following, we will only deal with SLT grammars G having the property from Lemma 1. For $i \in [4]$, we denote with $G(i)$ the SLT grammar (without start nonterminal) consisting of all productions of G of type (i) from Lemma 1.

A *straight-line program* (SLP) can be seen as a 1-SLT grammar $G = (N, \Sigma, S, P)$ containing only productions of the form $A(y) \to B(C(y))$ and $A(y) \to \sigma(y)$ with $B, C \in N$ and $\sigma \in \Sigma$. Thus, G contains ordinary rules of a context-free string grammar in Chomsky normal form (but written as monadic trees). Intuitively, if $\mathsf{val}(G) = a_1(\cdots a_n(y)\cdots)$ then G produces the string $a_1 \cdots a_n$ and we also write $\mathsf{val}(G) = a_1 \cdots a_n$. For a string $w = a_1 \cdots a_n$ and two numbers $l, r \in [n]$ with $l \leq r$ we denote by $w[l, r]$ the substring $a_l a_{l+1} \cdots a_r$. The following result is well-known, see e.g. [12].

Lemma 2. *For a given* SLP G *and two binary encoded numbers $l, r \in [|\mathsf{val}(G)|]$ with $l \leq r$ one can compute in polynomial time an* SLP G' *such that* $\mathsf{val}(G') = \mathsf{val}(G)[l, r]$.

3 Isomorphism of Rooted Unordered SLT-Compressed Trees

Let us fix an alphabet Σ. For $t \in T_\Sigma$ we denote with $\mathsf{uo}(t)$ the unordered rooted version of t. It is the node-labeled directed graph (V, E, λ) where $V = D(t)$ is the set of nodes, $E = \{(u, u \cdot i) \mid i \in \mathbb{N}, u \in \mathbb{N}^*, u \cdot i \in D(t)\}$ is the edge relation, and λ is the node-labelling function with $\lambda(u) = t[u]$. For an SLT grammar G, we also write $\mathsf{val}_{\mathsf{uo}}(G)$ for $\mathsf{uo}(\mathsf{val}(G))$.

For reasons that will become clear in a moment we have to restrict in this section to *ranked* trees, i.e., trees $t \in T_\Sigma$ such that for all $u, v \in D(t)$, if $t[u] = t[v]$ then u and v have the same number of children (nodes with the same label have the same number of children). For the purpose of deciding the isomorphism problem for unordered SLT-represented trees this is not a real restriction. Denote for a tree $t \in T_\Sigma$ the ranked tree $\mathsf{ranked}(t)$ such that $D(t) = D(\mathsf{ranked}(t))$ and for every $u \in D(t)$ with $t[u] = \sigma$: if u has k children in t, then $\mathsf{ranked}(t)[u] = \sigma_k$, where σ_k is a new symbol. Clearly, $\mathsf{uo}(s)$ and $\mathsf{uo}(t)$ are isomorphic if and only if $\mathsf{uo}(\mathsf{ranked}(s))$ and $\mathsf{uo}(\mathsf{ranked}(t))$ are isomorphic. Moreover, for an SLT grammar G we construct in polynomial time the SLT grammar $\mathsf{ranked}(G)$ obtained from G by changing every production $A \to t$ into $A \to \mathsf{ranked}(t)$, where ranked is extended to trees over Σ and nonterminals by defining $\mathsf{ranked}(t)[u] = t[u]$ if $t[u]$ is a nonterminal. Then $\mathsf{val}(\mathsf{ranked}(G)) = \mathsf{ranked}(\mathsf{val}(G))$ holds. Hence, in the following we will only consider ranked trees, and all SLT grammars will produce ranked trees as well.

For a tree $t \in T_\Sigma$ we denote by $\mathsf{dflr}(t) \in \Sigma^*$ its depth-first left-to-right traversal string. It is defined as $\mathsf{dflr}(\sigma(t_1, \ldots, t_k)) = \sigma\, \mathsf{dflr}(t_1) \cdots \mathsf{dflr}(t_k)$ for $\sigma \in \Sigma$, $k \geq 0$,

and $t_1, \ldots, t_k \in T_\Sigma$. Note that for ranked trees s and t it holds that: $\mathsf{dflr}(s) = \mathsf{dflr}(t)$ if and only if $s = t$. This is the reason for restricting to ranked trees: for unranked trees this equivalence fails. For instance, $\mathsf{dflr}((a(a(a))) = a^3 = \mathsf{dflr}(a(a, a))$.

Let $<_\Sigma$ be an order on Σ; it induces the *length-lexicographical ordering* $<_{\mathsf{llex}}$ on Σ by $u <_{\mathsf{llex}} v$ iff (*i*) $|u| < |v|$ or (*ii*) $|u| = |v|$ and there exist $p, u', v' \in \Sigma^*$ and $a, b \in \Sigma$ with $a <_\Sigma b$, $u = pau'$, and $v = pbv'$. We extend $<_{\mathsf{llex}}$ to T_Σ by $s <_{\mathsf{llex}} t$ iff $\mathsf{dflr}(s) <_{\mathsf{llex}} \mathsf{dflr}(t)$.

Statement (1) in the following lemma was shown in [4] by computing from G, H in polynomial time SLPs G', H' with $\mathsf{val}(G') = \mathsf{dflr}(\mathsf{val}(G))$ and $\mathsf{val}(H') = \mathsf{dflr}(\mathsf{val}(H))$. Equivalence of SLPs can be decided in polynomial time (this result was independently shown by Plandowski, Hirshfeld, Jerrum, Moller, and Mehlhorn, Sundar, Uhrig, see [12] for references). For statement (2) one can do binary search to find the first position where the string $\mathsf{val}(G')$ and $\mathsf{val}(H')$ differ.

Lemma 3. *Let G, H be* SLT *grammars. It is decidable in polynomial time whether or not (1) $\mathsf{val}(G) <_{\mathsf{llex}} \mathsf{val}(H)$ and (2) whether or not $\mathsf{val}(G) = \mathsf{val}(H)$.*

For a tree $t \in T_\Sigma$ we define its *canon* $\mathsf{canon}(t)$ as the smallest tree s w.r.t. $<_{\mathsf{llex}}$ such that $\mathsf{uo}(s)$ is isomorphic to $\mathsf{uo}(t)$. In order to determine $\mathsf{canon}(t)$ for $t = \sigma(t_1, \ldots, t_k)$ let $c_i = \mathsf{canon}(t_i)$ for $i \in [k]$ and let $c_{i_1} \leq_{\mathsf{llex}} c_{i_2} \leq_{\mathsf{llex}} \cdots \leq_{\mathsf{llex}} c_{i_k}$ be the length-lexicographically ordered list of canons $c_1, \ldots, c_k$. Then $\mathsf{canon}(t) = \sigma(c_{i_1}, \ldots, c_{i_n})$. The following lemma can be easily shown by an induction on the tree structure:

Lemma 4. *Let $s, t \in T_\Sigma$. Then $\mathsf{uo}(s)$ is isomorphic to $\mathsf{uo}(t)$ iff $\mathsf{canon}(s) = \mathsf{canon}(t)$.*

In the following, we denote a tree $A_1(A_2(\cdots A_n(t)\cdots))$, where $A_1, A_2, \ldots, A_n$ are unary nonterminals with $A_1A_2\cdots A_n(t)$.

Theorem 5. *From a given* SLT *grammar G one can construct in polynomial time an* SLT *grammar G' such that $\mathsf{val}(G') = \mathsf{canon}(\mathsf{val}(G))$.*

Proof. Let $G = (N, \Sigma, S, P)$. We assume that G contains no distinct nonterminals $A_1, A_2 \in N^{(0)}$ such that $\mathsf{val}_G(A_1) = \mathsf{val}_G(A_2)$. This is justified because we can test $\mathsf{val}_G(A_1) = \mathsf{val}_G(A_2)$ in polynomial time by Lemma 3 (and replace A_2 by A_1 in G in such a case). We will add polynomially many new nonterminals to G and change the productions for nonterminals from $N^{(0)}$ such that for the resulting SLT grammar G': $\mathsf{val}_{G'}(Z) = \mathsf{canon}(\mathsf{val}_G(Z))$ for every $Z \in N^{(0)}$.

Consider a nonterminal $Z \in N^{(0)}$ and let M be the set of all nonterminals in G that can be reached from Z. By induction, we can assume that G already satisfies $\mathsf{val}_G(A) = \mathsf{canon}(\mathsf{val}_G(A))$ for every $A \in M^{(0)} \setminus \{Z\}$. We distinguish two cases.

Case (*i*). Z is of type (1) from Lemma 1, i.e., has a production $Z \to \sigma(A_1, \ldots, A_k)$. Using Lemma 3 we construct an ordering $i_1, \ldots, i_k$ of $[k]$ such that $\mathsf{val}_G(A_{i_1}) \leq_{\mathsf{llex}} \mathsf{val}_G(A_{i_2}) \leq_{\mathsf{llex}} \cdots \leq_{\mathsf{llex}} \mathsf{val}_G(A_{i_k})$. We obtain G' by replacing the production $Z \to \sigma(A_1, \ldots, A_k)$ by $Z \to \sigma(A_{i_1}, \ldots, A_{i_k})$ and get $\mathsf{val}_{G'}(Z) = \mathsf{canon}(\mathsf{val}_G(Z))$.

Case (*ii*). Z is of type (2), i.e., has a production $Z \to B(A)$. Let $\{S_1, \ldots, S_m\} = M^{(0)} \setminus \{Z\}$ be an ordering such that $\mathsf{val}_G(S_1) <_{\mathsf{llex}} \mathsf{val}_G(S_2) <_{\mathsf{llex}} \cdots <_{\mathsf{llex}} \mathsf{val}_G(S_m)$. Note that A is one of these S_i. The sequence $S_1, S_2, \ldots, S_m$ partitions the set of all trees t in T_Σ into intervals $\mathcal{I}_0, \mathcal{I}_1, \ldots, \mathcal{I}_m$ with

– $\mathcal{I}_i = \{t \in T_\Sigma \mid \mathsf{val}_H(S_i) \leq_{\mathsf{llex}} t <_{\mathsf{llex}} \mathsf{val}_H(S_{i+1})\}$ for $1 \leq i \leq m-1$,
– $\mathcal{I}_0 = \{t \in T_\Sigma \mid t <_{\mathsf{llex}} \mathsf{val}_H(S_1)\}$, and $\mathcal{I}_m = \{t \in T_\Sigma \mid \mathsf{val}_H(S_m) \leq_{\mathsf{llex}} t\}$.

Consider the maximal $G(4)$-derivation $B(A) \Rightarrow^*_{G(4)} B_1B_2\cdots B_N(A)$ starting from $B(A)$, where B_i is a type-(3) nonterminal. Clearly, the number N might be of exponential size, but the set $\{B_1,\ldots,B_N\}$ can be easily constructed. In order to construct an SLT for $\mathsf{canon}(\mathsf{val}_G(Z))$, it remains to reorder the arguments in right-hand sides of the type-(3) nonterminals B_i. The problem is of course that different occurrences of a type-(3) nonterminal in the sequence $B_1B_2\cdots B_N$ have to be reordered in a different way. But we will show that the sequence $B_1B_2\cdots B_N$ can be split into $m+1$ blocks such that all occurrences of a type-(3) nonterminal in one of these blocks have to be reordered in the same way.

Let $t_k = \mathsf{val}_G(B_kB_{k+1}\cdots B_N(A))$ for $k \in [N]$ and $t_{N+1} = \mathsf{val}_G(A)$. Note that $t_1 = \mathsf{val}_G(Z) >_{\mathsf{llex}} \mathsf{val}_G(S_m)$ and that $t_{k+1} <_{\mathsf{llex}} t_k$ for all k. For $i \in [m]$ let k_i be the maximal position $k \leq N+1$ such that $t_k \geq_{\mathsf{llex}} \mathsf{val}_G(S_i)$. Since $t_1 \geq_{\mathsf{llex}} \mathsf{val}_G(S_m) \geq_{\mathsf{llex}} \mathsf{val}_G(S_i)$ this position is well defined. Note that if $A = S_i$, then $k_i = k_{i-1} = \cdots = k_1 = N+1$. For every $0 \leq i \leq m$, the interval $[k_{i+1}+1, k_i]$ is the set of all k such that $\mathsf{val}_G(t_k) \in \mathcal{I}_i$. Here we set $k_{m+1} = 0$ and $k_0 = N+1$. Clearly, the interval $[k_{i+1}+1, k_i]$ might be empty. The positions $k_0,\ldots,k_m$ can be computed in polynomial time using binary search combined with Lemma 3. To apply the latter, note that for a given k we can compute in polynomial time an SLT grammar for the tree t_k using Lemma 2 for the SLP consisting of all type-(4) productions that are used to derive $B_1B_2\cdots B_N$.

We now factorize the string $B_1B_2\cdots B_N$ as $B_1B_2\cdots B_N = u_mu_{m-1}\cdots u_0$, where $u_m = B_1\cdots B_{k_m-1}$ and $u_i = B_{k_{i+1}}\cdots B_{k_i-1}$ for $0 \leq i \leq m-1$. By Lemma 2 we can compute in polynomial time an SLP G_i for the string u_i. For the further consideration, we view G_i as a 1-SLT grammar consisting only of type-(4) productions. Note that $\mathsf{val}(G_i)$ is a linear tree, where every node is labelled with a type-(3) nonterminal. We now add reordered versions of type-(3) productions to G_i. Consider a type-(3) production $(C(y) \to \sigma(A_1,\ldots,A_j,y,A_{j+1},\ldots,A_k)) \in P$ where $C \in \{B_1,\ldots,B_N\}$. We add to G_i the type-(3) production $C(y) \to \sigma(A_{j_1},\ldots,A_{j_\nu},y,A_{j_{\nu+1}},\ldots,A_{j_k})$, where $\{j_1,\ldots,j_k\} = [k]$ and $0 \leq \nu \leq k$ are chosen such that

(1) $\mathsf{val}_G(A_{j_1}) \leq_{\mathsf{llex}} \mathsf{val}_G(A_{j_2}) \leq_{\mathsf{llex}} \cdots \leq_{\mathsf{llex}} \mathsf{val}_G(A_{j_k})$ and
(2) $\mathsf{val}_G(A_{j_\nu}) \leq_{\mathsf{llex}} \mathsf{val}_G(S_i) <_{\mathsf{llex}} \mathsf{val}_G(A_{j_{\nu+1}})$.

Note that if $\nu = k$ then condition (2) states that $\mathsf{val}_G(A_{j_k}) \leq_{\mathsf{llex}} \mathsf{val}_G(S_i)$, and if $\nu = 0$ then it states that $\mathsf{val}_G(S_i) <_{\mathsf{llex}} \mathsf{val}_G(A_{j_1})$. Also note that condition (2) ensures that for every tree $t \in \mathcal{I}_i$: $\mathsf{val}_G(A_{j_\nu}) \leq_{\mathsf{llex}} t <_{\mathsf{llex}} \mathsf{val}_G(A_{j_{\nu+1}})$. Hence, $\mathsf{val}_G(\sigma(A_{j_1},\ldots,A_{j_\nu},t, A_{j_{\nu+1}},\ldots,A_{j_k}))$ is a canon. The crucial observation now is that the above factorization $u_mu_{m-1}\cdots u_0$ of $B_1B_2\cdots B_N$ was defined in such a way that for every occurrence of a type-(3) nonterminal $C(y)$ in u_i, the parameter y will be substituted by a tree from $\mathcal{I}_i$ during the derivation from Z to $\mathsf{val}_G(Z)$. Hence, we reorder the arguments in the right-hand sides of nonterminal occurrences in u_i in the correct way to obtain a canon.

We now rename the nonterminals in the SLT grammars G_i (which are now of type-(3) and type-(4)) so that the nonterminal sets of $G, G_0,\ldots,G_m$ are pairwise disjoint. Let $X_i(y)$ be the start nonterminal of G_i after the renaming. Then we add to the current SLT grammar G the union of all the G_i, and replace the production $Z \to B(A)$ by

$Z \to X_m X_{m-1} \cdots X_0(A)$. The construction implies that $\mathsf{val}_{G'}(Z) = \mathsf{canon}(\mathsf{val}_G(Z))$ for the resulting grammar G'.

It remains to argue that the above construction can be carried out in polynomial time. All steps only need polynomial time in the size of the current SLT grammar. Hence, it suffices to show that the size of the SLT grammar is polynomially bounded. The algorithm is divided into $|N^{(0)}|$ many phases, where in each phase it enforces $\mathsf{val}_{G'}(Z) = \mathsf{canon}(\mathsf{val}_G(Z))$ for a single nonterminal Z. Consider a single phase, where $\mathsf{val}_{G'}(Z) = \mathsf{canon}(\mathsf{val}_G(Z))$ is enforced for a nonterminal Z. In this phase, we (*i*) change the production for Z and (*ii*) add new type-(3) and type-(4) productions to G (the union of the G_i above). But the number of these new productions is polynomially bounded in the size of the initial SLT grammar (the one before the first phase), because the nonterminals introduced in earlier phases are not relevant for the current phase. This implies that the additive size increase in each phase is bounded polynomially in the size of the initial grammar. □

Corollary 6. *The problem of deciding whether* $\mathsf{val}_{\mathsf{uo}}(G_1)$ *and* $\mathsf{val}_{\mathsf{uo}}(G_2)$ *are isomorphic for given* SLT *grammars* G_1 *and* G_2 *is* PTIME*-complete*

Proof. Membership in PTIME follows immediately from Lemma 3, Lemma 4, and Theorem 5. Moreover, PTIME-hardness already holds for dags, i.e., SLT grammars where all nonterminals have rank 0, as shown in [15]. □

4 Isomorphism of Unrooted Unordered SLT-Compressed Trees

In this section we show isomorphism for unrooted unordered trees represented by SLT grammars can be solved in polynomial time. An unrooted unordered tree t over Σ can be seen as a node-labeled (undirected) graph $t = (V, E, \lambda)$, where $E \subseteq V \times V$ is symmetric and $\lambda : V \to \Sigma$. Let $s = (V, E, \lambda)$ be a rooted unordered tree. The tree $\mathsf{ur}(s) = (V, E \cup E^{-1}, \lambda)$ is the unrooted version of s. An unrooted unordered tree t can be represented by an SLT grammar G by forgetting the order and root information present in G. Let $\mathsf{val}_{\mathsf{ur},\mathsf{uo}}(G) = \mathsf{ur}(\mathsf{uo}(\mathsf{val}(G)))$.

Let $t = (V, E, \lambda)$ be an unordered unrooted tree. For a node $v \in V$ we define the eccentricity $\mathsf{ecc}_t(v) = \max_{u \in V} \delta_t(u, v)$ and the diameter $\varnothing(t) = \max_{v \in V} \mathsf{ecc}_t(v)$, where $\delta_t(u, v)$ denotes the distance from u to v (i.e., the number of edges on the path from u to v in t). A node u of t is called *center node of* t if for all leaves v of t: $\delta_t(u, v) \leq (\varnothing(s) + 1)/2$. Let $\mathsf{center}(t)$ be the set of all center nodes of t. One can compute the center nodes by deleting all leaves of the tree and iterating this step, until the current tree consists of at most two nodes. These are the center nodes of t. In particular, t has either one or two center nodes. Another characterization of center nodes that is important for our algorithm is via longest paths. Let $p = (v_0, v_1, \ldots, v_n)$ be a longest simple path in t, i.e., $n = \varnothing(t)$. Then the middle points $v_{\lfloor n/2 \rfloor}$ and $v_{\lceil n/2 \rceil}$ (which are identical if n is even) are the center nodes of t and are independent of the concrete longest path p.

Note that there are two center nodes if and only if $\varnothing(t)$ is odd. Since our constructions are simpler if a unique center node exists, we first make sure that $\varnothing(t)$ is even. Let $\#$ be a new symbol not in Σ. For an unrooted unordered tree t we denote by $\mathsf{even}(t)$ the tree where every pair of edge $(u, v), (v, u)$ is replaced by the

edges $(u,v'),(v',v),(v,v'),(v',u)$, where v' is a new node labelled $\#$. Then for an SLT grammar $G = (N,\Sigma,P,S)$ we let $\mathsf{even}(G) = (N,\Sigma\cup\{\#\},P',S)$ be the SLT grammar where P' is obtained from P by replacing every subtree $\sigma(t_1,\ldots,t_k)$ with $\sigma\in\Sigma$, $k\geq 1$, in a right-hand side by the subtree $\sigma(\#(t_1),\ldots,\#(t_k))$. Observe that (*i*) $\mathsf{val}_{\mathsf{ur},\mathsf{uo}}(\mathsf{even}(G)) = \mathsf{even}(\mathsf{val}_{\mathsf{ur},\mathsf{uo}}(G))$, (*ii*) $\varnothing(\mathsf{even}(t)) = 2\cdot\varnothing(t)$ is even, i.e., $\mathsf{even}(t)$ has only one center node, and (*iii*) trees t and s are isomorphic if and only if $\mathsf{even}(t)$ and $\mathsf{even}(s)$ are isomorphic. Since $\mathsf{even}(G)$ can be constructed in polynomial time, we assume in the following that every SLT grammar produces a tree with a unique center node. For such a tree t we denote with $\mathsf{center}(t)$ its unique center node.

Let $u\in V$. The rooted version $\mathsf{root}(t,u)$ of t with root node u is $\mathsf{root}(t,u) = (V,E',\lambda)$, where $E' = \{(v,v')\in E \mid \delta_t(u,v) < \delta_t(u,v')\}$. Two unrooted unordered trees t_1,t_2 of even diameter are isomorphic iff $\mathsf{root}(t_1,\mathsf{center}(t_1))$ is isomorphic to $\mathsf{root}(t_2,\mathsf{center}(t_2))$. Thus, we can solve in polynomial time the isomorphism problem for unrooted unordered trees represented by SLT grammars G_1,G_2 by (*i*) computing for $i\in\{1,2\}$ in polynomial time a compressed representation $\tilde{u}_i$ of $u_i = \mathsf{center}(\mathsf{val}_{\mathsf{ur},\mathsf{uo}}(G_i))$ (Section 4.1), (*ii*) computing for $i\in\{1,2\}$ in polynomial time an SLT grammar G_i' such that $\mathsf{val}_{\mathsf{uo}}(G_i') = \mathsf{root}(\mathsf{val}_{\mathsf{ur},\mathsf{uo}}(G_i),u_i)$ (Section 4.2) and (*iii*) testing in polynomial time if $\mathsf{val}_{\mathsf{uo}}(G_1')$ is isomorphic to $\mathsf{val}_{\mathsf{uo}}(G_2')$ (Corollary 6).

4.1 Finding Center Nodes

Let $G = (N,\Sigma,S,P)$ be an SLT grammar. A *G-compressed path* p is a string of pairs $p = (A_1,u_1)\cdots(A_n,u_n)$ such that for all $i\in[n]$, $A_i\in N$, $A_1 = S$, $u_i\in D(t_i)$ is a Dewey address in t_i where $(A_i\to t_i)\in P$, $t_i[u_i] = A_{i+1}$ for $i<n$, and $t_i[u_n]\in\Sigma$. If we omit the condition $t_i[u_n]\in\Sigma$, then p is a partial G-compressed path. Note that by definition, $n\leq|N|$. A partial G-compressed path uniquely represents one particular node in the derivation tree of G, and a G-compressed path represents a leaf of the derivation tree and hence a node of $\mathsf{val}(G)$. We denote this node by $\mathsf{val}_G(p)$. The concatenation $u_1,u_2,\ldots,u_n$ of the Dewey addresses is denoted by $u(p)$.

For a context $t(y)\in\mathcal{C}_\Sigma$ we define $\mathsf{ecc}(t) = \mathsf{ecc}_t(y)$ (recall that in a context there is a unique occurrence of the parameter y) and $\mathsf{rty}(t) = \delta_t(\varepsilon,y)$ (the distance from the root to the parameter y). For a tree $s\in T_\Sigma$ we denote with $h(s)$ its height. We extend these notions to contexts $t\in\mathcal{C}_{\Sigma\cup N}$ and trees $s\in T_{\Sigma\cup N}$ by $\mathsf{ecc}(t) = \mathsf{ecc}(\mathsf{val}_G(t))$, $\mathsf{rty}(t) = \mathsf{rty}(\mathsf{val}_G(t))$, and $h(s) = h(\mathsf{val}_G(s))$. Eccentricity, distance from root to y, and height can be computed in polynomial time for SLT-represented trees bottom-up. To do so, observe that for contexts $t(y),t'(y)\in\mathcal{C}_{\Sigma\cup N}$ and a tree $s\in T_{\Sigma\cup N}$: $\mathsf{rty}(t[t']) = \mathsf{rty}(t)+\mathsf{rty}(t')$, $\mathsf{ecc}(t[t']) = \max\{\mathsf{ecc}(t'),\mathsf{ecc}(t)+\mathsf{rty}(t')\}$, and $h(t[s]) = \max\{h(s),\mathsf{rty}(t)+h(s)\}$. Similarly, for $t(y) = \sigma(s_1,\ldots s_i,y,s_{i+1},\ldots,s_k)\in\mathcal{C}_{\Sigma\cup N}$ and $s = \sigma(s_1,\ldots,s_k)\in T_{\Sigma\cup N}$: $\mathsf{rty}(t) = 1$, $\mathsf{ecc}(t) = 2+\max\{h(s_i)\mid i\in[k]\}$, and $h(s) = 1+\max\{h(s_i)\mid i\in[k]\}$.

Our search for the center node of an SLT-compressed tree is based on the following lemma. For a context $t(y)\in\mathcal{C}_\Sigma$, where u is the Dewey address of the parameter y, and a tree $s\in T_\Sigma$ we say that a node $v\in D(t[s])$ belongs to t if $v\in D(t)\setminus\{u\}$. Otherwise, we say that v belongs to s, which means that u is a prefix of v.

Lemma 7. *Let $t(y) \in \mathcal{C}_\Sigma$ be a context and $s \in T_\Sigma$ a tree such that $\varnothing(t[s])$ is even. Let $c = \mathsf{center}(t[s])$. Then c belongs to s if and only if $\mathsf{ecc}(t) \leq h(s)$.*

Lemma 8. *For a given* SLT *grammar G such that $\mathsf{val}_{\mathsf{ur},\mathsf{uo}}(G)$ has even diameter, one can construct a G-compressed path for $\mathsf{center}(\mathsf{val}_{\mathsf{ur},\mathsf{uo}}(G))$.*

Proof. Let $G = (N, \Sigma, S, P)$. Our algorithm for finding the center node for $\mathsf{val}(G)$ stores at each point of time a single tuple (t_l, A, t_r, p), where $t_l \in \mathcal{C}_{\Sigma\cup N}$ and $t_r \in T_{\Sigma\cup N} \cup \{\varepsilon\}$ are of polynomial size, $A \in N$, and p is a partial G-compressed path. It is started with the tuple $(y, S, \varepsilon, \varepsilon)$ The following invariants are preserved: If the current tuple is (t_l, A, t_r, p) is, then:

- If A has rank 0 then $t_r = \varepsilon$.
- $\mathsf{val}(G) = \mathsf{val}(t_l[A[t_r]])$ (here we set $t[\varepsilon] = t$).
- The tree $t_l[A[t_r]]$ can be derived from the start variable S.
- p is the partial G-compressed path to the distinguished A in $t_l[A[t_r]]$.
- $\mathsf{center}(\mathsf{val}_{\mathsf{ur},\mathsf{uo}}(G))$ belongs to the subcontext $\mathsf{val}(A)$ in $\mathsf{val}(t_l)[\mathsf{val}(A)[\mathsf{val}(t_r)]]$.

For the tuple (t_l, A, t_r, p), the algorithm distinguishes on the right-hand side of A. If this right-hand side has the form $A(B)$ or $A(B(y))$, then, by comparing $\mathsf{ecc}(t_l[B(y)])$ and $h(C[t_r])$ (we can compute these values in polynomial time, by constructing SLT grammars for $\mathsf{val}(t_l[B(y)])$ and $\mathsf{val}(C[t_r])$ and using the recursions for ecc, rty and h), we determine, whether the search for the center node has to continue in B or C, see Lemma 7. In the first case we continue with the tuple $(t_l, B, C[t_r], p \cdot (A, \varepsilon))$, and in the second case we continue with $(t_l[B(y)], C, t_r, p \cdot (A, 1))$.

Now assume that the right-hand side of A has the form $\sigma(A_1, \ldots, A_k)$. Let $t_i = t_l[\sigma(A_1, \ldots, A_{i-1}, y, A_{i+1}, \ldots, A_k)]$ for $i \in [k]$. Hence, $\mathsf{val}(G) = \mathsf{val}(t_i)[\mathsf{val}(A_i)]$. By comparing $\mathsf{ecc}(t_i)$ and $h(A_i)$ we want determine whether the center node belongs to $\mathsf{val}(t_i)$ or $\mathsf{val}(A_i)$, see Lemma 7. If the latter holds for some $i \in [k]$, we can continue the search in A_i, i.e., we continue with the tuple $(t_i, A_i, \varepsilon, p \cdot (A, i))$. On the other hand, assume that for all $i \in [k]$ the center point belongs to t_i. In particular, it does not belongs to any of the subtrees $\mathsf{val}(A_i)$. But by the last invariant, we known that the center point belongs to the subcontext $\mathsf{val}(A) = \sigma(\mathsf{val}(A_1), \ldots, \mathsf{val}(A_k))$. Hence, the σ-labelled node must be the center point and we can return its G-compressed path $p \cdot (A, \varepsilon)$. The case of a production $A(y) \to \sigma(A_1, \ldots A_{s-1}, y, A_{s+1}, \ldots, A_k)$ can be dealt with similarly. Note that $|t_l| + |t_r|$ stays bounded by the size of G. □

4.2 Re-Rooting of SLT Grammars

Let $G = (N, \Sigma, S, P)$ be an SLT grammar (as usual, having the normal form from Lemma 1) and p a G-compressed path. Let $s(p) \in T_{\Sigma\cup N}$ be the tree defined inductively as follows: Let $(A \to t) \in P$ and $u \in D(t)$. Then $s((A, u)) = t$. If $p = (A, t)p'$ with p' non-empty, then either (*i*) $u = \varepsilon$ and $t = B(C)$ or (*ii*) $u = i \in \mathbb{N}$ and $t[i] \in N^{(0)}$. In case (*i*) we set $s(p) = s(p')[C]$, in case (*ii*) we set $s(p) = t'[s(p')]$, where $t'(y)$ is obtained from t by replacing the i-th argument of the root by y. Note that $s(p') \in \mathcal{C}_{\Sigma\cup N}(\{y\})$ if p' starts with a nonterminal of rank 1. Let $s = s(p)$; its size is bounded by the size of G. Note that $s[u(p)]$ is a terminal symbol (recall that $u(p)$ denotes the

concatenation of the Dewey addresses in p). Assume that $s[u(p)] = \sigma \in \Sigma$. Let $\#$ be a fresh symbol and let s' be obtained from s by changing the label at $u(p)$ from σ to $\#$. Let $s' \Rightarrow_G^* s''$ be the shortest derivation such that $s''[\varepsilon] = \delta \in \Sigma$ (it consists of at most $|N|$ derivation steps). We denote the $\#$-labeled node in s'' by u. Finally, let t be obtained from s'' by changing the unique $\#$ into σ. We define the *p-expansion* of G, denoted $\mathsf{ex}_G(p)$, as the tuple (t, u, σ, δ). Note that $\mathsf{val}_G(p)$ is the unique $\#$-labelled node in $\mathsf{val}_G(s'')$. The p-expansion can be computed in polynomial time from G and p.

The p-expansion (t, u, σ, δ) has all information needed to construct a grammar G' representing the rooted version at p of $\mathsf{val}(G)$. If $u = \varepsilon$ then also $\mathsf{val}_G(p) = \varepsilon$. Since G is already rooted at ε nothing has to be done in this case and we return $G' = G$. If $u \neq \varepsilon$ then $\mathsf{val}_G(p) \neq \varepsilon$ and hence t contains two terminal nodes which uniquely represent the root node and the node $\mathsf{val}_G(p)$ of the tree $\mathsf{val}(G)$.

Let $s_1 \in T_\Sigma$ be a rooted ordered tree representing the unrooted unordered tree $\tilde{s}_1 = \mathsf{ur}(\mathsf{uo}(s_1))$. Let $u \neq \varepsilon$ be a node of s_1. Let $s_1[\varepsilon] = \delta \in \Sigma$ and $s_1[u] = \sigma \in \Sigma$. Since $u \neq \varepsilon$, we can write $s_1 = \delta(\zeta_1, \ldots, \zeta_{i-1}, t'[\sigma(\xi_1, \ldots, \xi_m)], \zeta_{i+1}, \ldots, \zeta_k)$, where t' is a context, and $u = iu'$, where u' is the Dewey address of the parameter y in t'. A rooted ordered tree s_2 that represents the rooted unordered tree $\tilde{s}_2 = \mathsf{root}(\tilde{s}_1, u)$ can be defined as $s_2 = \sigma(\xi_1, \ldots, \xi_m, \mathsf{rooty}(t')[\delta(\zeta_1, \ldots, \zeta_{i-1}, \zeta_{i+1}, \ldots, \zeta_k)])$, where rooty is a function mapping contexts to contexts defined recursively as follows ($f \in \Sigma$, $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_\ell \in T_\Sigma$, and $t(y), t'(y) \in \mathcal{C}_\Sigma$):

$$\mathsf{rooty}(y) = y \tag{1}$$

$$\mathsf{rooty}(f(t_1, \ldots, t_{i-1}, y, t_{i+1}, \ldots, t_\ell)) = f(t_1, \ldots, t_{i-1}, y, t_{i+1}, \ldots, t_\ell) \tag{2}$$

$$\mathsf{rooty}(t[t'(y)]) = \mathsf{rooty}(t')[\mathsf{rooty}(t(y))] \tag{3}$$

Intuitively, the mapping rooty unroots a context $t(y)$ towards its y-node u, i.e., it reverses the path from the root to u. Thus, for instance, $\mathsf{rooty}(f(a, y, b)) = f(a, y, b)$ and $\mathsf{rooty}(f(a, g(c, y, d), b)) = g(c, f(a, y, b), d)$.

Lemma 9. *From a given* SLT *grammar G and a G-compressed path p one can construct in polynomial time an* SLT *grammar G' such that $\mathsf{val}_{\mathsf{uo}}(G')$ is isomorphic to $\mathsf{root}(\mathsf{val}_{\mathsf{ur,uo}}(G), \mathsf{val}_G(p))$.*

Proof. Let $G = (N, \Sigma, S, P)$ and $\mathsf{ex}_G(p) = (t, u, \sigma, \delta)$. If $u = \varepsilon$ then define $G' = G$. If $u \neq \varepsilon$ then we can write $t = \delta(B_1, \ldots, B_{i-1}, t'[\sigma(\xi_1, \ldots, \xi_m)], B_{i+1}, \ldots, B_k)$, where $B_j \in N^{(0)}$, $\xi_j \in T_N$, t' is a context composed of nonterminals $A \in N^{(1)}$ and contexts $f(\zeta_1, \ldots, \zeta_{j-1}, y, \zeta_{j+1}, \ldots, \zeta_l)$ ($f \in \Sigma$, $\zeta_j \in T_N$), and $u = iu'$, where u' is the Dewey address of the parameter y in t'.

We define $G' = (N \uplus N', \Sigma, S, P')$ where $N' = \{A' \mid A \in N^{(1)}\}$. To define the production set P', we extend the definition of rooty to contexts from $\mathcal{C}_{\Sigma \cup N}$ by (*i*) allowing in the trees t_j from Equation (2) also nonterminals, and (*ii*) defining for every $B \in N^{(1)}$, $\mathsf{rooty}(B(y)) = B'(y)$. We now define the set of productions P' of P as follows: We put all productions from P except for the start production $(S \to s) \in P$ into P'. For the start variable S we add to P' the production

$$S \to \sigma(\xi_1, \ldots, \xi_m, \mathsf{rooty}(t')[\delta(B_1, \ldots, B_{i-1}, B_{i+1}, \ldots, B_k)]). \tag{4}$$

Moreover, let $A \in N^{(1)}$ and $(A(y) \to \zeta) \in P$. If this is a type-(3) production, then we add $A'(y) \to \zeta$ to P'. If $\zeta = B(C(y))$ then add $A'(y) \to C'(B'(y))$ to P'.

A simple induction shows that $\mathsf{val}_{G'}(A') = \mathsf{rooty}(\mathsf{val}_G(A))$ for every $A \in N^{(1)}$. This implies that $\mathsf{val}_{G'}(\mathsf{rooty}(c(y))) = \mathsf{rooty}(\mathsf{val}_G(c(y)))$ for every context $c(y)$ that is composed of contexts $f(\zeta_1, \ldots, \zeta_{j-1}, y, \zeta_{j+1}, \ldots, \zeta_l)$ ($\zeta_j \in T_N$) and nonterminals $A \in N^{(1)}$. In particular, $\mathsf{val}_{G'}(\mathsf{rooty}(t')) = \mathsf{rooty}(\mathsf{val}_G(t'(y)))$ for the context t'. This, and the form of the start production of G' (4) easily imply that $\mathsf{val}_{\mathsf{uo}}(G')$ is isomorphic to $\mathsf{root}(\mathsf{val}_{\mathsf{ur},\mathsf{uo}}(G), \mathsf{val}_G(p))$. □

Corollary 10. *The problem of deciding whether* $\mathsf{val}_{\mathsf{ur},\mathsf{uo}}(G_1)$ *and* $\mathsf{val}_{\mathsf{ur},\mathsf{uo}}(G_2)$ *are isomorphic for given* SLT *grammars* G_1 *and* G_2 *is* PTIME*-complete.*

Proof. The upper bound follows from Lemma 8, Lemma 9, and Corollary 6. Hardness for PTIME follows from the PTIME-hardness for dags [15] and the fact that isomorphism of rooted unordered trees can be reduced to isomorphism of unrooted unordered trees by labelling the roots with a fresh symbol. □

5 Further Results

Bisimulation on SLT-compressed trees. Fix a set Σ of node labels. Let $G = (V, E, \lambda)$ be a directed node-labelled graph, i.e., $E \subseteq V \times V$ and $\lambda : V \to \Sigma$. A binary relation $R \subseteq V \times V$ is a bisimulation on G, if for all $(u, v) \in R$ the following three conditions hold: (*i*) $\lambda(u) = \lambda(v)$, (*ii*) if $(u, u') \in E$ then there exists $v' \in V$ such that $(v, v') \in E$ and $(u', v') \in R$, and (*iii*) if $(v, v') \in E$ then there exists $u' \in V$ such that $(u, u') \in E$ and $(u', v') \in R$. Let the relation $\sim$ be the union of all bisimulations on G. It is the largest bisimulation and an equivalence relation. Two rooted unordered trees s, t with node labels from Σ and roots r_s, r_t are bisimulation equivalent if $r_s \sim r_t$ holds in the disjoint union of s and t. For instance, $f(a, a, a)$ and $f(a, a)$ are bisimulation equivalent but $f(g(a), g(b))$ and $f(g(a, b))$ are not. For a rooted unordered tree t we define the bisimulation canon $\mathsf{bcanon}(t)$ inductively: Let $t = f(t_1, \ldots, t_n)$ ($n \geq 0$) and let $b_i = \mathsf{bcanon}(t_i)$. Then $\mathsf{bcanon}(t) = f(s_1, \ldots, s_m)$, where (*i*) for every $i \in [m]$, s_i is isomorphic to one of the b_j, and (*ii*) for every $i \in [n]$ there is a unique $j \in [m]$ such that s_i and b_j are isomorphic as rooted unordered trees. In other words: Bottom-up, we eliminate repeated subtrees among the children of a node. For instance, $\mathsf{bcanon}(f(a, a, a)) = f(a) = \mathsf{bcanon}(f(a, a))$. Induction on the height of trees shows:

Lemma 11. *Let* s *and* t *be rooted unordered trees. Then* s *and* t *are bisimulation equivalent if and only if* $\mathsf{bcanon}(s)$ *and* $\mathsf{bcanon}(t)$ *are isomorphic.*

The proof of the following theorem is similar to those of Theorem 5.

Theorem 12. *From a given* SLT *grammar* G *one can compute a new* SLT *grammar* G' *such that* $\mathsf{val}_{\mathsf{uo}}(G')$ *is isomorphic to* $\mathsf{bcanon}(\mathsf{val}_{\mathsf{uo}}(G))$.

From Corollary 6, Lemma 11, and Theorem 12 we get:

Corollary 13. *For given* SLT *grammars* G_1 *and* G_2 *one can check in polynomial time, whether* $\mathsf{val}_{\mathsf{uo}}(G_1)$ *and* $\mathsf{val}_{\mathsf{uo}}(G_2)$ *are bisimulation equivalent.*

Non-linear ST grammars. Recall from Section 2 that ST grammars are exponentially more succinct than SLT grammars. So, the following should not be surprising:

Lemma 14. *A given* ST *grammar can be transformed in exponential time into an equivalent* SLT *grammar.*

Using this lemma, the upper bounds in the following statement follow from Corollary 6, 10, and 13. For the lower bound, one can reduce from QBF using gadgets from [9].

Theorem 15. *The following questions are* PSPACE*-hard and in* EXPTIME *for given* ST *grammars G_1 and G_2:*

- *Are $\mathsf{val}_{\mathsf{uo}}(G_1)$ and $\mathsf{val}_{\mathsf{uo}}(G_2)$ isomorphic (resp., bisimulation equivalent)?*
- *Are $\mathsf{val}_{\mathsf{ur,uo}}(G_1)$ and $\mathsf{val}_{\mathsf{ur,uo}}(G_2)$ isomorphic?*

The precise complexity of these questions remains open. Since an ST grammar can be transformed into a hierarchical graph definition for a dag, we rediscover the following result from [3]: Bisimulation equivalence for dags given by hierarchical graph definitions is PSPACE-hard and in EXPTIME.

References

1. A. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison–Wesley, Reading, MA, 1974.
2. J. Balcázar, J. Gabarró, and M. Sántha. Deciding bisimilarity is P-complete. *Formal Aspects of Computing*, 4:638–648, 1992.
3. R. Brenguier, S. Göller, and O. Sankur. A comparison of succinctly represented finite-state systems. In *Proc. CONCUR 2012*, LNCS 7454, pages 147–161. Springer, 2012.
4. G. Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML document trees. *Inf. Syst.*, 33(4–5):456–474, 2008.
5. S. R. Buss. Alogtime algorithms for tree isomorphism, comparison, and canonization. In *Kurt Gödel Colloquium 97*, pages 18–33, 1997.
6. M. Charikar, E. Lehman, A. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Trans. Inf. Theory*, 51(7):2554–2576, 2005.
7. B. Das, P. Scharpfenecker, and J. Torán. Succinct encodings of graph isomorphism. In *Proc. LATA 2014*, LNCS 8370, pages 285–296. Springer, 2014.
8. H. Galperin and A. Wigderson. Succinct representations of graphs. *Inf. Contr.*, 56:183-198, 1983.
9. B. Jenner, J. Köbler, P. McKenzie, and J. Torán. Completeness results for graph isomorphism. *J. Comput. Syst. Sci.*, 66(3):549–566, 2003.
10. T. Lengauer and K. W. Wagner. The correlation between the complexities of the nonhierarchical and hierarchical versions of graph problems. *J. Comput. Syst. Sci.*, 44:63–93, 1992.
11. S. Lindell. A logspace algorithm for tree canonization (extended abstract). In *Proc. STOC'92*, pages 400–404. ACM, 1992.
12. M. Lohrey. Algorithmics on SLP-compressed strings: a survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.
13. M. Lohrey, S. Maneth, and F. Peternek. Compressed tree canonization. arXiv.org, 2015. http://arxiv.org/abs/1502.04625
14. M. Lohrey, S. Maneth, and M. Schmidt-Schauß. Parameter reduction and automata evaluation for grammar-compressed trees. *J. Comput. Syst. Sci.*, 78(5):1651–1669, 2012.
15. M. Lohrey and C. Mathissen. Isomorphism of regular trees and words. *Inf. Comput.*, 224:71–105, 2013.